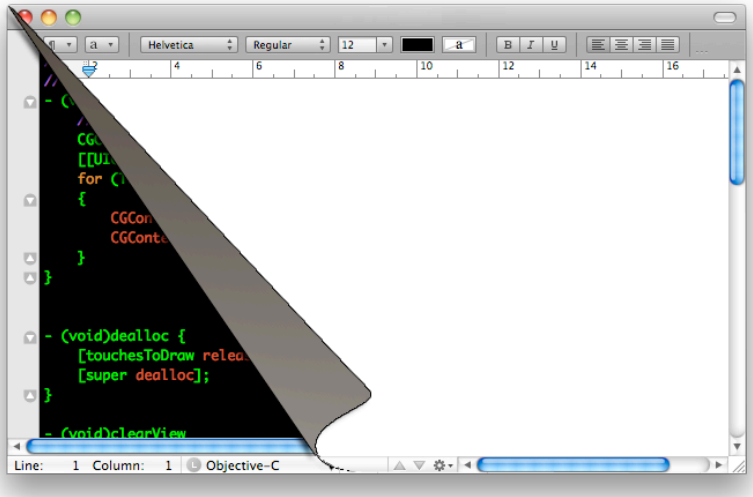


Designing Interactive Systems II

Computer Science Graduate Program SS 2011

Prof. Dr. Jan Borchers
Media Computing Group
RWTH Aachen University

http://hci.rwth-aachen.de/dis2

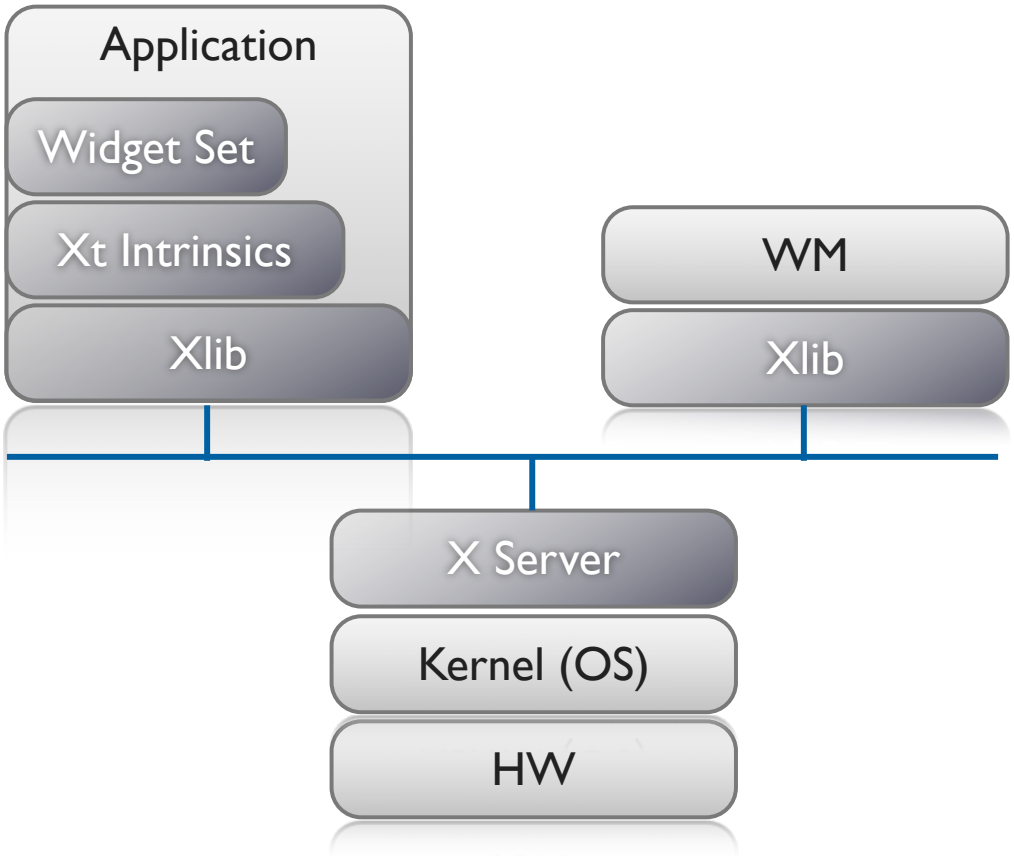
Review: WM, UITK

- What are the main responsibilities of the window manager?
- Name some UI elements the window manager provides.
- What is late refinement?
- What is a UITK?
- Static and dynamic widget hierarchy?

The X Window System

- Asente, Reid (Stanford): W window system for V OS, (1982)
  - W moved BWS&GEL to remote machine, replaced local library calls with synch. communication
  - Simplified porting to new architectures, but slow under Unix
- MIT: X as improvement over W (1984)
  - Asynchronous calls: much-improved performance
  - Application = client, calls X Library (Xlib) which packages and sends GEL calls to the X Server and receives events using the X Protocol.
  - Similar to Andrew, but window manager separate
  - X10 first public release, X11 cross-platform redesign

X: Architecture



UITK+WM

BWS+GEL

X is close to our 4-layer architecture model

X Server

- X11 ISO standard, but limited since static protocol
- X server process combines GEL and BWS
  - Responsible for one keyboard (one EL), but n physical screens (GLs)
  - One machine can run several X servers
- Applications (with UITK) and WM are clients
- GEL: Direct drawing, raster model, rectangular clipping
  - X-Server layers:
    - Top = Device-independent X (DIX)
    - Bottom = Device-dependent X (DDX)
  - BWS can optionally buffer output regions

X Protocol

- Between X server process and X clients (incl. WM)
- Asynchronous, bidirectional byte stream, order guaranteed by transport layer
  - Implemented in TCP, but also others (DECnet,...)
  - Creates about 20% time overhead with apps over network
- Four packet types
  - Request (Client→Server)
  - Reply, Event, Error (Server→Client)
- Packets contain opcode, length, and sequence of resource IDs or numbers

Typical Xlib application (pseudocode)

```
#include Xlib.h, Xutil.h
Display *d; int screen; GC gc; Window w; XEvent e;
main () {
   d = XOpenDisplay(171.64.77.1:0);
   screen = DefaultScreen(d);
   w = XCreateSimpleWindow(d, DefaultRootWindow(d), x,y,w,h,
     border, BlackPixel(d), WhitePixel(d)); // foreground &
     background
   XMapWindow(d, w);
   gc = XCreateGC(d, w, mask, attributes); // Graphics Context
     setup left out here
   XSelectInput(d, w, ExposureMask|ButtonPressMask);
   while (TRUE) {
     XNextEvent(d, &e);
       switch (e.type) {
          case Expose: XDrawLine (d, w, gc, x,y,w,h); break;
          case ButtonPress: exit(0);
} } }
```


X: Resources

- Logical: pixmap, window, graphic context, color map, visual (graphics capabilities), font, cursor
- Real: setup (connection), screen (several), client
- All resources identified via RIDs
- Events: as in reference model; from user, BWS, and apps, piped into appropriate connection
- X Server is simple single-entrance server (round-robin), user-level process

Window Manager

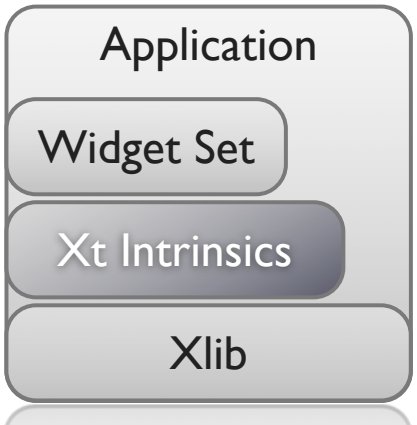
- Ordinary client to the BWS
- Communicates with apps via *hints* in X Server
- Look&Feel mechanisms are separated from Look&Feel policy
- Late refinement (session, user, application, call)

Window Manager

- Dynamically exchangeable, even during session
  - twm, ctwm, gwm, mwm (Motif), olwm (OpenLook), rtl (Tiling), ...
  - Implement different policies for window & icon placement, appearance, all without static menu bar, mostly pop-ups, flexible listener modes
- No desktop functionality (separate app)
- Only manages windows directly on background (root) window, rest managed by applications (since they don't own root window space)

X: UITK

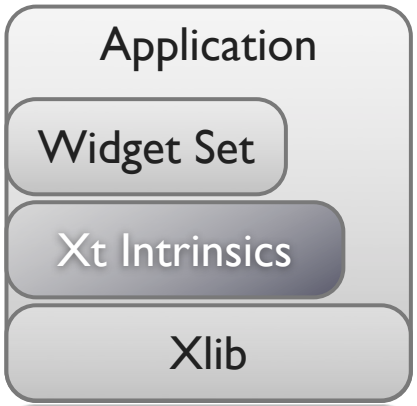
- X programming support consists of 3 layers
- Xlib:
  - Lowest level, implements X protocol client, procedural (C)
  - Programming on the level of the BWS
  - Hides networking, but not X server differences (see "Visual")
  - Packages requests, usually not waiting for reply (asynchronous)
  - At each Xlib call, checks for events from server and creates queue on client (access with XGetNextEvent())
  - Extensions require changing Xlib & Xserver source & protocol



X: UITK

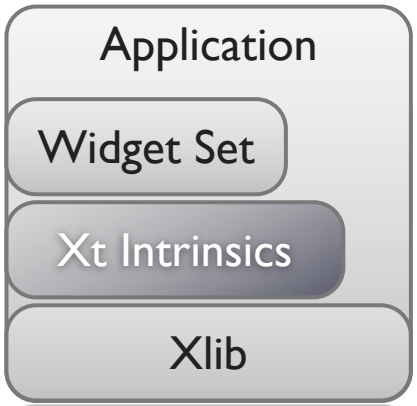
- Xlib offers functions to create, delete, and modify server resources (pixmaps, windows, graphic contexts, color maps, visuals, fonts), but app has to do resource composition
- Display (server connection) is parameter in most calls

- X Toolkit Intrinsics (Xt)
  - Functions to implement an OO widget set class (static) hierarchy
  - Programming library and runtime system handling widgets
  - Exchangeable (InterViews/C++), but standard is in C
  - Each widget defined as set of "resources" (attributes) (XtNborderColor,...)



X: UITK

- X Toolkit Intrinsics
  - Just abstract meta widget classes (Simple, Container, Shell)
  - At runtime, widgets have 4 states
    - Created (data structure exists, linked into widget tree, no window)
    - Managed (Size and position have been determined—policy)
    - Realized (window has been allocated in server; happens automatically for all children of a container)
    - Mapped (rendered on screen)—may still be covered by other window!



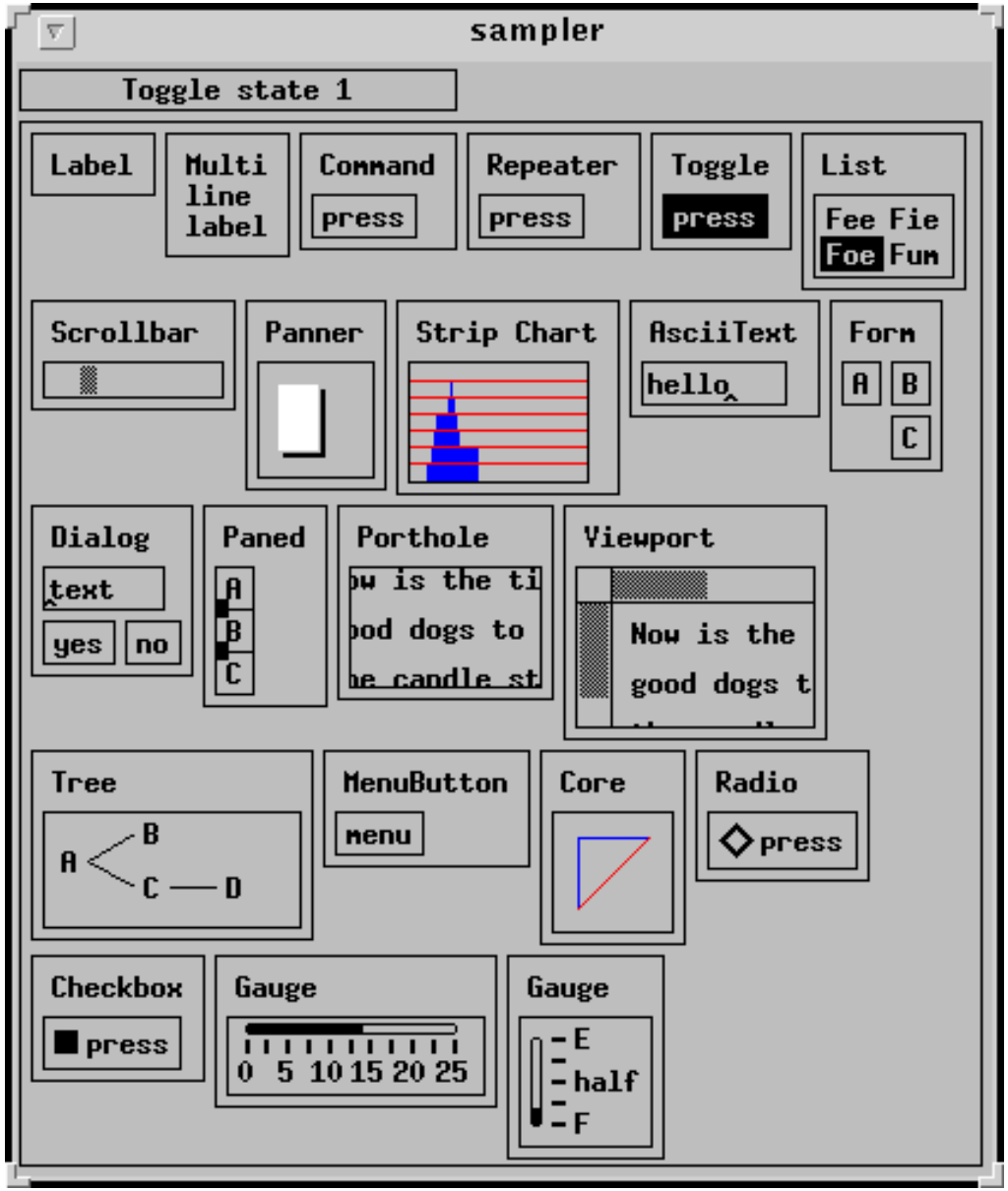
UITK

- X Toolkit Intrinsics (continued)
  - Xt Functions (XtRealizeWidget(),...) are generic to work with all widget classes
  - Event dispatch:
    - Defined for most events in translation tables (I→A) in Xt
    - →Widgets handle events alone (no event loop in app)!
    - App logic in callback functions registered with widgets

Widget Sets

- Collection of user interface components
- Together with WM, define look&feel of system
- Several different ones available for X
  - Athena (original, simple widget set, ca. 20 widgets, 2-D, no strong associated style guide) — Xaw... prefix
  - Motif (Open Software Foundation, commercial, 2.5-D widget set, >40 widgets, industry standard for X, comes with style guide and UIL)—Xm... prefix
- Programming model already given in Intrinsics
  - Motif just offers convenience functions

Athena Widget Set



- Original, free, extensible
- Ugly, simple
- Class hierarchy:
  - *Simple* — Base class for all other Athena widgets. Does nothing, but adds new resources such as cursor and border pixmap.

Athena

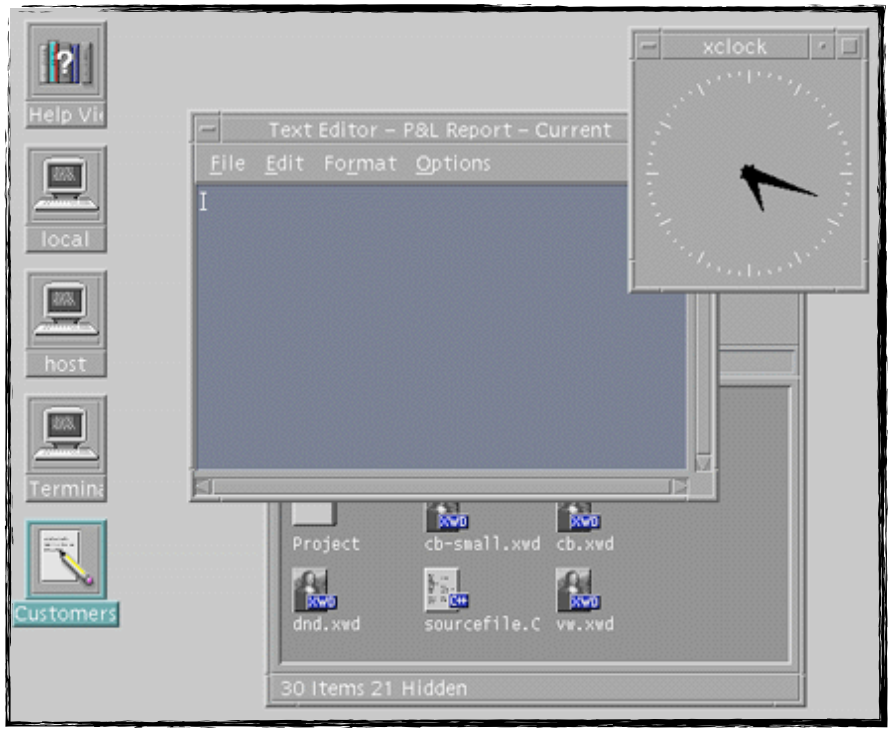
- Standard widgets:
  - *Label* Draws text and/or a bitmap.
  - *Command* Momentary push-button
  - *Toggle* Push-button with two states.
  - *MenuButton* Push-button that brings up a menu.
  - *Grip* Small widget used to adjust borders in a Paned widget.
  - *List* Widget to allow user to select one string from a list.
  - *Scrollbar* Widget to allow user to set a value; typically to scroll another widget.
  - *Box* Composite widget which simply lays children out left-to-right.
  - *Form* Constraint widget which positions children relative to each other.
  - *Dialog* Form widget for dialog boxes.
  - *Paned* Constraint widget letting user adjust borders between child widgets.

  - *Text* Base class for all other text classes.
  - *TextSink* Base class for other text sinks.
  - *TextSrc* Base class for other text sources (subclasses for ASCII and multi-byte text)

  - *SimpleMenu* Shell which manages a simple menu.
  - *Sme* RectObj which contains a simple menu entry (blank).
  - *SmeBSB* Menu entry with a string and optional left & right bitmaps.
  - *SmeLine* Menu entry that draws a seperator line.

Athena

- Special widgets:
  - *Repeater* Command that repeatedly calls its associated callback function for as long as it's held.
  - *Panner* Widget to allow user to scroll in two dimensions.
  - *StripChart* Widget to display a scrolling graph.
  - *Porthole* Composite widget which allows a larger widget to be windowed within a smaller window. Often controlled by Panners.
  - *Viewport* Constraint widget, like a Porthole with scrollbars.
  - *Tree* Constraint widget, lays its children out in a tree.



What Is Motif?

- Style Guide (book) for application developer
- Widget set (software library) implementing Style Guide
- Window Manager (mwm)
- UIL (User Interface Language)

The Motif Widget Set

- Simple Widgets: XmPrimitive
  - XmLabel, XmText, XmSeparator, XmScrollbar,...
- Shell Widgets: Shell
  - Widgets talking to Window Manager (root window children)
  - Application shells, popup shells,...
- Constraint Widgets: XmManager
  - Containters like XmDrawingArea, XmRowColumn,...
  - Complex widgets like XmFileSelectionBox,...

Programming with Motif

- Initialize Intrinsics
  - Connect to server, allocate toolkit resources
- Create widgets
  - Building the dynamic widget tree for application
  - Tell Intrinsics to manage each widget
- Realize widgets
  - Sensitize for input, per default also make visible (map)
- Register callbacks
  - Specify what app function to call when widgets are triggered
- Event loop
  - Just call Intrinsics (XtMainLoop()) – app ends in some callback!

hello.c: A Simple Example

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xlib.h>
#include <Xm/Xm.h>
#include <Xm/PushB.h>

void ExitCB (Widget w, caddr_t client_data, XmAnyCallbackStruct
*call_data)
{
  XtCloseDisplay (XtDisplay (w));
  exit (0);
}

void main(int argc, char *argv[])
{
  Widget toplevel, pushbutton;

  toplevel = XtInitialize (argv [0], "Hello", NULL, 0, &argc, argv);
  pushbutton = XmCreatePushButton (toplevel, "pushbutton", NULL, 0);
  XtManageChild (pushbutton);

  XtAddCallback (pushbutton, XmNactivateCallback, (void *) ExitCB,
NULL);

  XtRealizeWidget (toplevel);
  XtMainLoop ();
}
```


Resource files in X

- Where does the title for the PushButton come from?
- → Resource file specifies settings for application
- Syntax: Application.PathToWidget.Attribute: Value
- Resource Manager reads and merges several resource files (system-, app- and user-specific) at startup (with priorities as discussed in reference model)

File "Hello":

```
Hello.pushbutton.labelString:   Hello World
Hello.pushbutton.width:         100
Hello.pushbutton.height:        20
```


User Interface Language UIL

- Resource files specify late refinement of widget attributes, but cannot add widgets
- Idea: specify actual widget tree of an application outside C source code, in UIL text file
  - C source code only contains application-specific callbacks, and simple stub for user interface
  - UIL text file is translated with separate compiler
  - At runtime, Motif Resource Manager reads compiled UIL file to construct dynamic widget tree for app
- Advantage: UI clearly separated from app code Decouples development

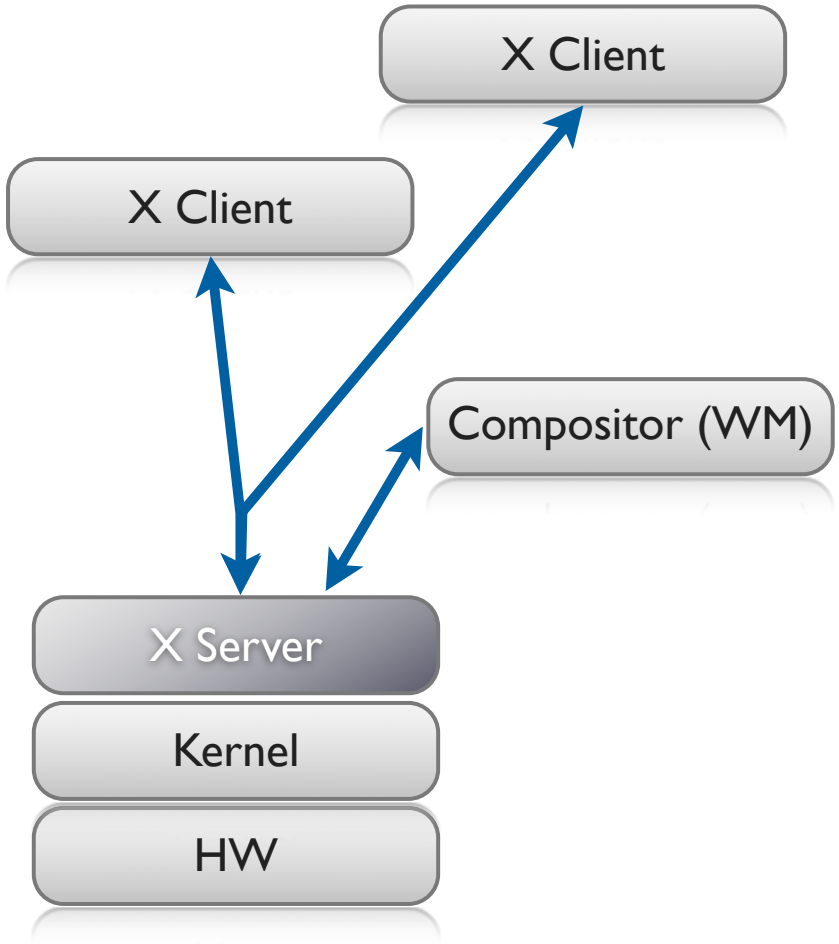
Wayland: Motivation

- A lot of functionality was moved from the X Server to the kernel
- An X server has to support a large amount of functionality
  - Core fonts (code tables, glyph rasterization, XLFDs)
- Rendering pipeline designed in the 1980s
- WMs add lots of decoration and transforms to windows
- No network transparency

Wayland is...

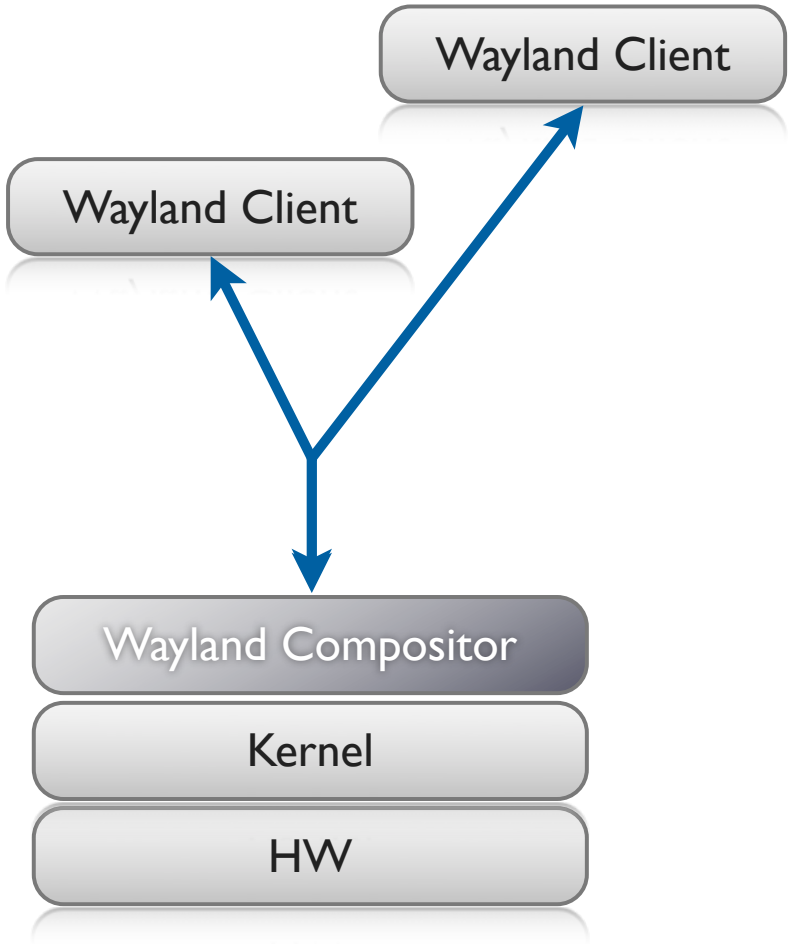
- A communication protocol between the compositor and its clients (similar to Xlib)
- An implementation of that protocol as a C library

Architecture: X



- Kernel passes events from the hardware to the X Server
- X Server determines window to receive event
- Client reacts to event and returns rendering request
- Compositor recomposites screen
- X Server renders

Architecture: Wayland



- Kernel passes events from the hardware to the compositor
- Check scenegraph to determine which window receives the event
- Client reacts to event and renders UI
- Compositor recomposites screen

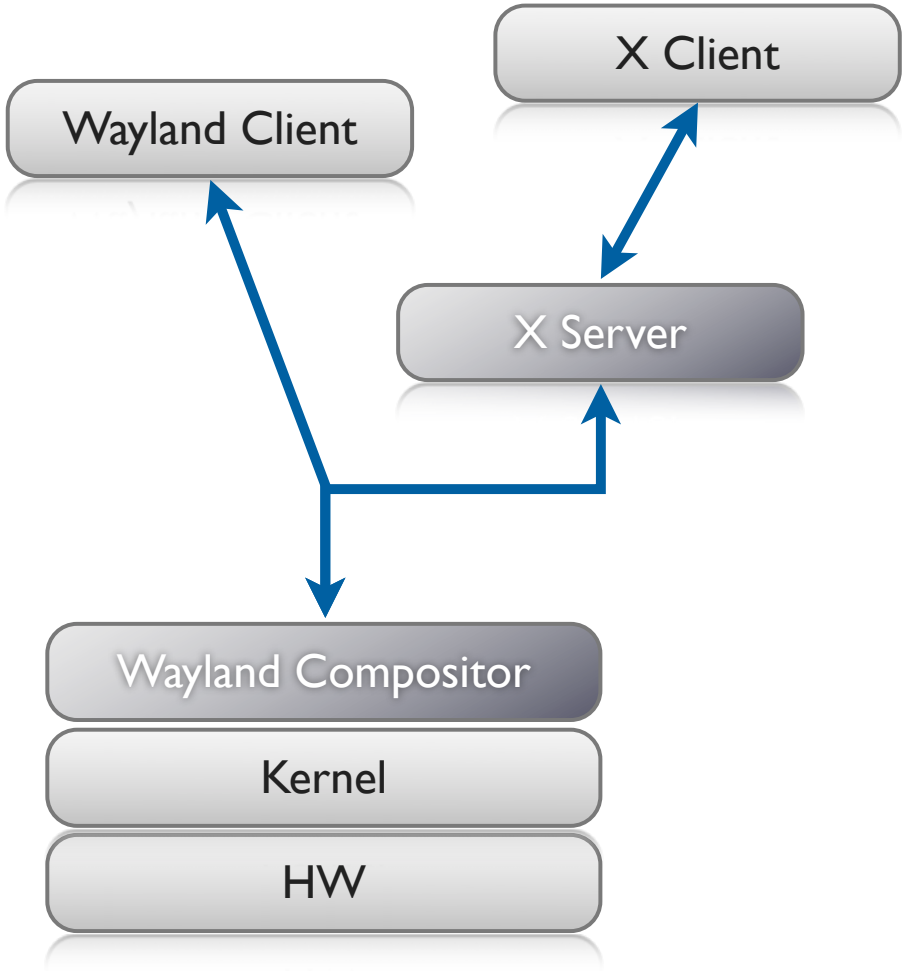
Wayland Rendering

- Direct rendering mechanism (DRI2)
  - Already used in current X servers
- Client and server share a video memory buffer
- Application renders into buffer (using, e.g., OpenGL)
- Compositor uses this buffer as texture

Wayland: Display Updates

- Using two or more buffers
  - Render content in a new buffer
  - Tell the compositor to use that new buffer as texture
- Using one buffer
  - Requires synchronization: avoid race between rendering and compositor
  - New content rendered into back buffer and copied to global buffer

X as Wayland Client



- Provide backward compatibility path
- Only small changes to X server required
- X server passes root window or top-level windows
- Wayland handles presentation of the windows

Wayland: UITK Support